

Dynamic Coupling Measurement for Aspect-Oriented Software

Haihao Shen, Hanyue Yang, Jianjun Zhao
School of Software
Shanghai Jiao Tong University
800 Dongchuan Road, Shanghai 200240, China
{haihaoshen, diezhongdie, zhao-jj}@sjtu.edu.cn

ABSTRACT

Coupling indicates the strength of association established by a connection from one module to another. Aspect-oriented programming (AOP) is a new technique to support separation of concerns in software development. In aspect-oriented (AO) software, a common way to define and measure coupling for AO software is through structural properties and static code analysis. Most of the existing measures and assessment frameworks concentrate on the static coupling evaluation of AO software. However, because of dynamic binding and the common presence of unused aspect code, the resulting coupling measures are imprecise as they do not perfectly reflect the actual coupling taking place among aspects and/or classes at runtime.

How coupling can be defined and precisely measured based on dynamic analysis of AO systems are presented in this paper. This paper focuses on dynamic coupling measurement for AO software. Firstly, a coupling framework for AO systems is specially designed to dynamically count the dependencies between aspects and classes in the systems. Secondly, based on this framework, a coupling measures suite for dynamic assessing the coupling in AO software is proposed. Thirdly, the mathematical properties of these measures are discussed to show that measures satisfy the properties, which a good coupling measure should have. Next, implementation issues for a dynamic coupling measures collection tool, called AJDCM, are described. Lastly, empirical evaluation of dynamic coupling measures is performed.

1. INTRODUCTION

Coupling is the measure of the strength of association established by a connection from one module to another [8]. High quality software design, among many other principles, should obey the principles of low coupling. The increasing importance has been placed on coupling measurement for evaluating and predicting the quality of software.

Aspect-oriented software development (AOSD), adding the concept of an aspect, is an emerging technique to support separation of concerns in software development. An aspect is a new first class entity that encapsulates crosscutting behavior in an application. Aspect-oriented programming (AOP) is a research area that has gained an increasing amount of attention lately due to its offering the idea of a new modular unit that encapsulates crosscutting concerns which would otherwise be scattered across multiple modules.

In aspect-oriented (AO) programs, the basic unit is an aspect or class. An aspect with its encapsulation of states (attributes) with associated modules such as advices, pointcuts, and methods (operations) is a significantly different abstraction in comparison to

the class within object-oriented (OO) software. Thus, in AO systems, the coupling is mainly about the degree of interdependence among aspects and classes. Moreover, in order to better measure the coupling of AO systems, different types of interactions should be considered between aspects and classes in the system.

As the broad use of OO design and programming matures in industry, coupling measurement for OO software has been studied extensively. However, since AO systems introduce new features, coupling measures cannot be directly applied to AO measurements. Despite the growing body of work dedicated to static measure coupling in AO systems [6, 7, 9, 15], there is no research related to dynamic coupling measurement. Because of dynamic crosscutting, polymorphism, dynamic bindings and the common presence of unused (“dead”) code in AO software, the static coupling measures are imprecise as they do not perfectly reflect the actual coupling taking place at runtime.

The goals of this paper are: (i) to create a standard terminology for fundamental structural elements in AO systems, (ii) to propose a unified dynamic coupling framework for measuring AO programs, (iii) to formally define a suite of dynamic coupling measures that can be collected through a dynamic analysis of the code by executing the code and saving information at runtime, (iv) to systematically analyze mathematical properties of dynamic coupling measures, (v) to empirically evaluate dynamic coupling measures.

The rest of the paper is organized as follows. Section 2 briefly introduces the background knowledge of AOP in AspectJ. Section 3 distinguishes different types of dynamic coupling and presents a dynamic coupling framework for defining coupling measures for AO systems. Section 4 defines a terminology for an AO system. Section 5 proposes 12 dynamic coupling measures based on the framework presented in Section 3 and studies the mathematical properties of the proposed coupling measure suite. Section 6 describes the implementation issues to collect dynamic coupling data. Section 7 offers the empirical evaluation of dynamic coupling measures. Section 8 discusses some problems in current research. Conclusion remarks and future work are presented in Section 9.

2. AOP AND ASPECTJ

The core idea of AOP is called *weaving* that means in case one of join points is reached, control flow changes into certain points in the execution of an application.

As a representative of the most well-known families of available AOP languages, AspectJ [10] is a seamless extension to Java. An AspectJ program can be divided into two parts: *base code* which includes classes, interfaces and other standard Java constructs, *aspect code* which implements the crosscutting concerns in the program. The execution of base code might trigger the execution of aspect code leading to coupling relationships between classes and aspects

which are not transparent. In this paper, AspectJ is adopted as target language to show the unique features by adding some new concepts and associated constructs, which include join points, pointcuts, advices and aspect.

- A *join point* constitutes the infrastructure of crosscutting concerns. A join point is a well-defined point in the execution of a program, such as a method call or access to a class member, etc.
- A *pointcut* consists of logically combined *pointcut designators* representing the different kinds of join points that exist. AspectJ provides the means to build hierarchies of pointcut expressions.
- An *advice* defines code that executes when a pointcut is reached. It is a method-like mechanism which consists of three kinds: *before*, *after* and *around advice*.
- An *aspect* is a modular unit of crosscutting implementation. Each aspect encapsulates functionality that crosscuts other classes in a program. Aspects can contain pointcuts, advices and methods.

All of these features provide the basis of the research on dynamic coupling measurement for AO software.

3. DYNAMIC COUPLING FRAMEWORK

Most of coupling frameworks deal with static analysis of coupling dependencies of the system. Due to the new introduced AspectJ features, dynamic behaviors identified at runtime, influence the coupling measurements. Therefore, in order to define dynamic coupling measures for AO software, a dynamic coupling framework will be proposed, which is based on frameworks for static coupling measurement in AO software [6, 7, 15]. This paper offers three decision criteria that need to be considered when classifying dynamic coupling measures, which are granularity, type of connection, locus of impact. Each of the six criteria will be discussed.

3.1 Granularity

Granularity refers to the domain entity of the measure, i.e., what components are to be measured. The framework focuses on coupling caused by dependencies that occur between aspects and/or classes in an AO system. Here, we consider the coupling dependence between aspects and/or classes.

3.2 Type of Connection

The type of a coupling connection between two modules is determined by the mechanism that is used to establish the coupling connection. A coupling connection between a client item and a server item is defined in [8]. A server and client item can be associated with a server and client class respectively. The client item uses the server item as a service. For instance, a method A calls another method B, which is a mechanism that leads to coupling connection between the client item, method A, and the server item, method B.

The work carried out by Bartsch [7] gave an overview of five kinds of coupling types including static and dynamic coupling features. Two groups of them are typical dynamic coupling connection. In this paper, join point based coupling and invocation based coupling will be discussed.

3.2.1 Join Point Based Coupling

Differing from any coupling connection in OO systems, join point based coupling describes coupling between an aspect and another class/aspect due to the weaving mechanism of advice code

```

1: public class C1 {
2:     C2 c2;
3:     public C1() {
4:         c2 = new C2();
5:         public static void main(String[] args) {
6:             C1 c1 = new C1();
7:             c1.m1();
8:             c1.m4();
9:         }
10:     public void m1() {
11:         c2.m2();
12:     }
13:     public void m4() {
14:         c2.m2();
15:     }
16: }
17: public class C2 {
18:     public void m2() {}
19: }
20: public class C3 {
21:     public void m3() {}
22: }
23: public aspect A1 {
24:     private C3 c3;
25:     public void am1() { A2.aspectOf().am2(); }
26:     pointcut pc1(Object c): cflow(execution(* m1()))
27:         && (within(C1) || within(C2)) && this(c);
28:     pointcut pc2(): adviceexecution() && within(A2);
29:     before(Object c): pc1(c) {
30:         am1();
31:         c3 = new C3();
32:         c3.m3();
33:     }
34:     after(): pc2() {
35:         A2.aspectOf().am2();
36:         A2.aspectOf().am2();
37:     }
38: }
39: public aspect A2 {
40:     private C3 c3;
41:     public void am2() {}
42:     pointcut pc3(): call(* m2());
43:     before(): pc3() {
44:         c3 = new C3();
45:         c3.m3();
46:     }
47: }

```

Figure 1: A sample AspectJ program

into base code through pointcuts picking up specifically join points. Most of the connections related to join points can be statically inferred. However, some pointcuts refer to dynamic states and it is only possible to determine if the join point is going to be picked up at runtime. In AspectJ, the following pointcuts refer to dynamic properties: *cflow*, *cflowbelow*, *this*, *target*, *args* and *if* (except for *if (true)* and *if (false)*). Join point based coupling just exhibits a coupling connection between the class/aspect associated with the join points and the aspect via a control flow change at runtime. In the connection of join point based coupling, the client item is the woven advice; the client class is the aspect containing the advice; the server item is the join point expression picked out and the server class is the class of the currently object executing the join point.

Figure 1 shows a sample AspectJ program. The program contains three classes C1, C2, C3 and two aspects A1, A2. The aspect A1 has one method am1 and two pieces of advice related to pointcuts pc1, pc2 respectively. pc1, containing *cflow* designator, is a dynamic pointcut. This control flow change is invisible. All information about this coupling can be revealed at runtime. pc2, containing *adviceexecution* designator, creates a coupling connection between two aspects. The aspect A2 has one method am2 and one piece of advice related to pointcuts pc3. Through the rest of the paper, this example program is used as a small working example to demonstrate idea of dynamic coupling measurement for AO software.

Owing to aspect A1 containing dynamic pointcut *cflow* pc1, when the program is executed, all join points are selected from which the control flow change originates, then aspect A1 and A2

Table 1: Join Point Based Coupling Sets of The Working Example

#	Join Point Mechanism	Client Class	Client Item	Server Item	Server Class
1	method execution join point	aspect A1	advice a1 with pointcut pc1	execution(C1.m1())	class C1
2	attribute reference join point	aspect A1	advice a1 with pointcut pc1	get(C2.C1.c2)	class C1
3	method call join point	aspect A2	advice a3 with pointcut pc3	call(C2.m2())	class C1
4	advice execution join point	aspect A1	advice a2 with pointcut pc2	adviceexecution(A2.before)	aspect A2
5	method call join point	aspect A1	advice a1 with pointcut pc1	call(C2.m2())	class C1
6	method execution join point	aspect A1	advice a1 with pointcut pc1	execution(C2.m2())	class C2
7	method call join point	aspect A2	advice a3 with pointcut pc3	call(C2.m2())	class C1
8	advice execution join point	aspect A1	advice a2 with pointcut pc2	adviceexecution(A2.before)	aspect A2

Table 2: Invocation Based Coupling Sets of the Working Example

# of Join Point	Line of Code	Client Class	Client Item	Server Item	Server Class
1	21	aspect A1	method am1	method am2	aspect A2
1	28	aspect A1	advice a1 with pointcut pc1	method m3	class C3
2	21	aspect A1	method am1	method am2	aspect A2
2	28	aspect A1	advice a1 with pointcut pc1	method m3	class C3
3	39	aspect A2	advice a3 with pointcut pc3	method m3	class C3
4	30	aspect A1	advice a2 with pointcut pc2	method am2	aspect A2
4	31	aspect A1	advice a2 with pointcut pc2	method am2	aspect A2
5	21	aspect A1	method am1	method am2	aspect A2
5	28	aspect A1	advice a1 with pointcut pc1	method m3	class C3
6	21	aspect A1	method am1	method am2	aspect A2
6	28	aspect A1	advice a1 with pointcut pc1	method m3	class C3
7	39	aspect A2	advice a3 with pointcut pc3	method m3	class C3
8	30	aspect A1	advice a2 with pointcut pc2	method am2	aspect A2
8	31	aspect A1	advice a2 with pointcut pc2	method am2	aspect A2

will be triggered separately. Table 1 shows all join point based coupling connections when Figure 1 the working program runs. The first column shows the sequence of selected join points at runtime. The second column names the type of join point. The latter four columns represent that advice of some aspect crosscuts a class whose executing object reaches some join point dynamically. Note that the fifth column is the join point expression. In Table 1, #4 shows that the join point is reached when *before* advice of aspect A2 executes, *after* advice of aspect A1 crosscuts aspect A2. Note that only advice execution join point type can achieve association between two aspects, other join point types lead association between an aspect and a class.

3.2.2 Invocation Based Coupling

Similar to coupling based on invocation in OO program, method or advice of an aspect sends messages to methods of a class or an aspect. Within this runtime session, messages sent from (received by) one object to (from) other objects, which brings a method invocation. For example, a method m1 or an advice a1 of aspect A calls method m2 of class C or another aspect B, which is a mechanism that leads to coupling connection between the client item, which is method m1 or advice a1, and the server item, which is method m2. Correspondingly, the client class is aspect A and the server class is class C or aspect B.

Table 2 shows all invocation based coupling connections when Figure 1 the working program runs. An invocation based coupling is described by a source aspect and method/advice sending the message, a line of code, and a target entity and method. Corresponding to column# of Table 1, the first column denotes definite join point reached at runtime. The second column shows the line of code. The latter four columns represent that a method or an advice of some aspect calls a method of a class or another aspect.

3.3 Locus of Impact

If an aspect is used in a coupling connection, a distinction made between export and import coupling is important. The client-server

relationship introduces import and export coupling. Export coupling means that the aspect is the server class of the connection. Import coupling means that the aspect is the client class of the connection. Export and import coupling define a counting rule for a certain coupling connection.

For join point based coupling, export coupling counts the number of aspects that are coupled to a given class/aspect. Import coupling counts the number of classes and aspects that are coupled to a given aspect. This is because a class is perceived as the server class that export join points which can be used by aspects that act as the client class.

For invocation based coupling, export coupling counts the number of aspects that make calls to a method of a given class/aspect. Import coupling counts the number of classes and aspects whose methods would be called by a given aspect. This is because a class is perceived as the server class whose methods are called by aspects that act as the client class.

4. TERMINOLOGY

In order to formally define dynamic coupling measures in AO programs precisely and unambiguously, the basic terminology sets should be defined on which to build measures definitions using set theory and first order logic. The terminology is derived from a similar terminology in [5, 15] for OO systems.

4.1 Sets

The first step is to define the basic sets in AO systems.

- C: Sets of classes in the system. Note that interfaces can be handled similarly with classes, so they are also included in C.
- A: Sets of aspects in the system.
- COM: Sets of components in the system. COM can be partitioned into the subsets of classes (C) and aspects (A), $COM = C \cup A$.

- **M**: Sets of methods in the system. M are composed of a subset of aspects' methods and a subset of classes' methods. $M(C)$ represents methods of all classes. For each class $c \in C$, $M(c)$ is the set of class c 's methods. $M(A)$ represents methods of all aspects. For each aspect $a \in A$, $M(a)$ is the set of aspect a 's methods.
- **ADV**: Sets of advices in the system. $ADV(A)$ represents advices of all aspects. For each aspect $a \in A$, $ADV(a)$ is the set of aspect a 's advices.
- **MOD**: Sets of modules of all aspects in the system. In an AO system, an aspect contains several types of modules, i.e., *advice*, *intertype declaration*, *pointcut* and *method*. Only *advices* and *methods* are involved in MOD set for the consideration of dynamic coupling measurement issues in this paper. For each aspect $a \in A$, $MOD(a) = ADV(a) \cup M(a)$.
- **JP**: Sets of join point expressions at runtime in the system.
- **HC**: Sets of hashcode of join point at runtime in the system. For each execution, $hashcode \in HC$ remains equally and identifies a unique join point expression $jp \in JP$.
- **N**: Sets of lines of code in the system. The elements of this set are all natural numbers.

4.2 Relations

Mathematical relations on the sets are introduced that are fundamental to the definitions of the measures.

- **CR** is the set of dynamic crosscutting that occur at runtime in the system: $CR \subseteq ADV \times A \times HC \times JP \times COM$. The five tuples indicates that when some component $com \in COM$ executing picks up join point $jp \in JP$ identified by $hashcode \in HC$, corresponding advice $adv \in ADV$ of some aspect $a \in A$ will be triggered to crosscut this component dynamically.
- **ME** is the set of actual passing messages in the system: $ME \subseteq MOD \times A \times HC \times N \times M \times COM$. The six tuples indicates that when some join point whose identifier is $hashcode \in HC$ is reached, a line of code $n \in N$, a source aspect $a \in A$ and module $mod \in MOD$ send the message to a target component $com \in COM$ and method $m \in M$.
- **OI** is the set of possible method invocations in the system. In this paper, **OI** is divided into two groups: explicit invocation and implicit invocation. Explicit invocation **EOI** is caused by dynamic messages. Implicit invocation **IOI** is caused by dynamic crosscutting. Now the definition of these two sets will be given. $EOI \subseteq MOD \times A \times M \times COM$, an explicit invocation is characterized by the invoking aspect $a \in A$ and module $mod \in MOD$ and another aspect/class $com \in COM$ and method $m \in M$ being invoked. $IOI \subseteq ADV \times A \times JP \times COM$, an implicit invocation represents that some component $com \in COM$ executing reaches join point $jp \in JP$, corresponding advice $adv \in ADV$ of some aspect $a \in A$ is triggered to crosscut this component.

Here, a small working example, as shown in Figure 1, will be used to illustrate the definitions above. Note that join points identified by $hc3$, $hc5$, $hc7$ respectively have the same expression $call(C2.m2())$, so join point expression elements of the tuples are all $jp3$ in **CR**.

$CR = \{(a1, A1, hc1, jp1, C1), (a1, A1, hc2, jp2, C1), (a3, A2, hc3, jp3, C1), (a2, A1, hc4, jp4, A2), (a1, A1, hc5, jp3, C1), (a1, A1, hc6, jp6, C2), (a3, A2, hc7, jp3, C1), (a2, A1, hc8, jp4, A2)\}$.

$ME = \{(am1, A1, hc1, 21, am2, A2), (a1, A1, hc1, 28, m3, C3), (am1, A1, hc2, 21, am2, A2), (a1, A1, hc2, 28, m3, C3), (a3, A2, hc3, 39, m3, C3), (a2, A1, hc4, 30, am2, A2), (a2, A1, hc4, 31, am2, A2), (am1, A1, hc5, 21, am2, A2), (a1, A1, hc5, 28, m3, C3), (am1, A1, hc6, 21, am2, A2), (a1, A1, hc6, 28, m3, C3), (a3, A2, hc7, 39, m3, C3), (a2, A1, hc8, 30, am2, A2), (a2, A1, hc8, 31, am2, A2)\}$.
 $EOI = \{(am1, A1, am2, A2), (a1, A1, m3, C3), (a3, A2, m3, C3), (a2, A1, am2, A2)\}$.
 $IOI = \{(a1, A1, jp1, C1), (a1, A1, jp2, C1), (a3, A2, jp3, C1), (a2, A1, jp4, A2), (a1, A1, jp3, C1), (a1, A1, jp6, C2)\}$.

5. DYNAMIC COUPLING MEASURES

In this section, 12 dynamic coupling measures are defined and mathematical properties of the proposed suite is also studied.

5.1 A Dynamic Coupling Metrics Suite

Based on dynamic coupling framework presented in Section 3 and the terminology defined in Section 4, six kinds of dynamic coupling measures are provided. The measures are all defined as cardinalities of specific sets.

- **Dynamic Crosscutting**. Within a runtime session, it is possible to count the total number of times that aspects crosscut present classes/aspects. Two crosscuts are considered to be the same if the advice, aspect, hashcode, join point expression, component are the same. The condition reflects the fact that a unique control flow change means to imply a different crosscuts. As shown in Table 1, there exists 8 dynamic crosscutting, including 6 times of aspect $A1$ crosscuts $C1, C2, A2$.
- **Distinct Implicit Method Invocations**. A simpler alternative is to count the number of distinct implicit method invocation caused by dynamic crosscutting. For example, when an advice of some aspect crosscuts a recursive method of a class, the number of dynamic crosscutting is the same as the times of recursion, accordingly, only once distinct implicit method invocation for which the join point expressions are the same.
- **Distinct Implicit Components**. It is also possible to count only the number of distinct server classes. As shown in Table 1, aspect $A1$ crosscuts $C1, C2, A2$, so the measure of this set is 3.
- **Dynamic Messages**. Within a runtime session, it is possible to count the total number of distinct messages sent from (received by) one object to (from) other objects. Two messages are considered to be the same if their source and target classes/aspects, the method invoked in the target component, the statement from which it is invoked in the source aspect are the same. The condition reflects the fact that a different context of invocation is considered to imply a different message. As shown in Table 2, aspect $A1$ sends 12 distinct messages to other classes/aspects.
- **Distinct Explicit Method Invocations**. An alternative is to count the number of distinct method invoked by each advice in each aspect. As shown in Table 2, there are 3 distinct explicit method invocations: $\{(am1, A1, am2, A2), (a1, A1, m3, C3), (a2, A1, am2, A2)\}$.
- **Distinct Explicit Components**. It is to count only the number of distinct server classes that a method in a given object uses (is used by). As shown in Table 2, there are 3 distinct explicit method invocations: $\{(am1, A1, A2), (a1, A1, C3), (a2, A1, A2)\}$.

Table 3: Summary of Dynamic Coupling Measures for AO Software

Locus of Impact	Type of Connection	Measures	Set Definition
Import Coupling	Join point based coupling	Dynamic Crosscutting	$ IC_DC(a) = \{(adv, a, hc, jp, com) adv \in ADV, a \in A, hc \in HC, jp \in JP, com \in COM, a \neq com, (adv, a, hc, jp, com) \in CR\} $
		Distinct Implicit Method Invocations	$ IC_II(a) = \{(adv, a, jp, com) adv \in ADV, a \in A, jp \in JP, com \in COM, a \neq com, (adv, a, hc, jp, com) \in CR, (adv, a, jp, com) \in IOI\} $
		Distinct Implicit Components	$ IC_IC(a) = \{(adv, a, com) adv \in ADV, a \in A, com \in COM, a \neq com, (adv, a, hc, jp, com) \in CR\} $
	Invocation based coupling	Dynamic Messages	$ IC_DM(a) = \{(mod, a, hc, n, m, com) mod \in MOD, a \in A, hc \in HC, n \in N, m \in M, com \in COM, a \neq com, (mod, a, hc, n, m, com) \in ME\} $
		Distinct Explicit Method Invocations	$ IC_EI(a) = \{(mod, a, m, com) mod \in MOD, a \in A, m \in M, com \in COM, a \neq com, (mod, a, hc, n, m, com) \in ME, (mod, a, m, com) \in EOI\} $
		Distinct Explicit Components	$ IC_EC(a) = \{(mod, a, com) mod \in MOD, a \in A, com \in COM, a \neq com, (mod, a, hc, n, m, com) \in ME\} $
Export Coupling	Join point based coupling	Dynamic Crosscutting	$ EC_DC(com) = \{(adv, a, hc, jp, com) adv \in ADV, a \in A, hc \in HC, jp \in JP, com \in COM, a \neq com, (adv, a, hc, jp, com) \in CR\} $
		Distinct Implicit Method Invocations	$ EC_II(com) = \{(adv, a, jp, com) adv \in ADV, a \in A, jp \in JP, com \in COM, a \neq com, (adv, a, hc, jp, com) \in CR, (adv, a, jp, com) \in IOI\} $
		Distinct Implicit Components	$ EC_IC(com) = \{(adv, a, com) adv \in ADV, a \in A, com \in COM, a \neq com, (adv, a, hc, jp, com) \in CR\} $
	Invocation Based Coupling	Dynamic Messages	$ EC_DM(com) = \{(mod, a, hc, n, m, com) mod \in MOD, a \in A, hc \in HC, n \in N, m \in M, com \in COM, a \neq com, (mod, a, hc, n, m, com) \in ME\} $
		Distinct Explicit Method Invocations	$ EC_EI(com) = \{(mod, a, m, com) mod \in MOD, a \in A, m \in M, com \in COM, a \neq com, (mod, a, hc, n, m, com) \in ME, (mod, a, m, com) \in EOI\} $
		Distinct Explicit Components	$ EC_EC(com) = \{(mod, a, com) mod \in MOD, a \in A, com \in COM, a \neq com, (mod, a, hc, n, m, com) \in ME\} $

Table 3 shows the formal set definitions of the dynamic coupling measures when the granularity is the aspect and class. The measures definition are differentiated *import* from *export* coupling, that is the locus of impact for an aspect or a class. They are also differentiated *join point based coupling* from *invocation based coupling*, that is the type of connection for dynamic coupling between an aspect and a class/aspect. As shown in Table 3, (I/E)C_DC focuses on CR relation; (I/E)C_II focuses on IOI relation; (I/E)C_DM focuses on ME relation; (I/E)C_EI focuses on EOI relation. An intuitive textual explanation only for the first set: IC_DC(a). Other sets can be interpreted in a similar manner.

IC_DC(a): A set containing all tuples (*advice*, *aspect*, *hashcode of join point*, *join point expression*, *component*) such that there exists dynamic crosscutting that when executing a component (aspect), a join point expression captured, which identified by its unique hashcode, then the advice of aspect is triggered to crosscut the component. The corresponding metric is simply the cardinality of this set. Note that the aspect must be different from the component ($a \neq com$), because we are focusing on dependencies that contribute to coupling between aspects and classes/aspects, not cohesion relation between different classes or the same aspect.

Return to the working example in Figure 1, the result of dynamic coupling measurement is provided in Table 4 by using the measures shown in Table 3.

5.2 Mathematical Properties

Five mathematical properties of coupling measures for OO software was presented in [8]. It is thought that the mathematical properties are also remarkable for AOP systems. The five mathematical properties of dynamic coupling measures will be studied in this subsection. The motivation of discussing measures' mathematical properties is to perform an initial theoretical validation by demonstrating that the measures have intuitive properties that can be justified. Let S be an AO system, $JPCoupling(S)$ be all dynamic coupling measures sets caused by join point based coupling, $INCoupling(S)$ be all dynamic coupling measures sets whose type of connection is invocation based coupling.

- **Nonnegativity.** Because the dynamic coupling measures measure the cardinality of sets, it is not possible for the dynamic

Table 4: Results of Dynamic Coupling Measures of the Working Example

	A1	A2	C1	C2	C3
IC_DC	6	2	0	0	0
IC_II	5	1	0	0	0
IC_IC	3	1	0	0	0
IC_DM	12	2	0	0	0
IC_EI	3	1	0	0	0
IC_EC	3	1	0	0	0
EC_DC	0	2	5	1	0
EC_II	0	1	4	1	0
EC_IC	0	1	2	1	0
EC_DM	0	8	0	0	6
EC_EI	0	2	0	0	2
EC_EC	0	2	0	0	2

coupling measures to be negative, e.g., IC_DC returns a set of tuples (adv, a, hc, jp, com) $\in ADV \times A \times HC \times JP \times COM$.

- **Null values.** All sets of join point based coupling measures are empty if and only if the set of dynamic crosscutting is empty: $CR = \emptyset \Leftrightarrow JPCoupling(S) = \emptyset$. All sets of invocation based coupling measures are empty if and only if the set of dynamic messages is empty: $ME = \emptyset \Leftrightarrow INCoupling(S) = \emptyset$.
- **Monotonicity.** Let $a \in A$ be an aspect of S . Some relationships were added to a to form a new aspect a' . Let S' be the AO system which is identical to S except that aspect a is replaced by a' . Then $JPCoupling(a) \leq JPCoupling(a')$, $JPCoupling(S) \leq JPCoupling(S')$. The property of monotonicity states that if a relationship between an aspect and a class/aspect is added into an AO system, coupling of the system can be decrease impossibly.
- **Impact of merging classes.** Let $a1, a2 \in A$ be two aspects of S . Let a' be the aspect which is the union of $a1$ and $a2$. Let S' be the AO system which is identical to S except that $a1$ and $a2$ are replaced by a' . Then $JPCoupling(a1) + JPCoupling(a2)$

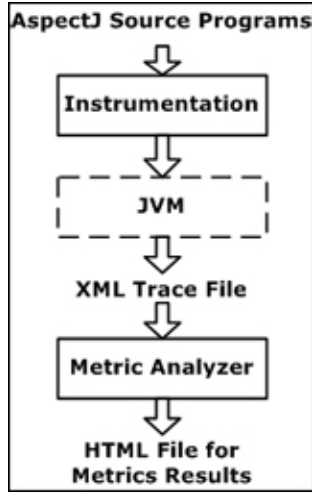


Figure 2: Architecture of the tool

$\geq \text{JPCoupling}(a')$, $\text{JPCoupling}(S) \geq \text{JPCoupling}(S')$. Merging of aspects may induce that relationships disappear. For example, after the tuples of the form $(am1, A1, am2, A2)$ are transformed into tuples of the form $(am1, A', am2, A')$, the coupling connection between $A1$ and $A2$ disappeared. The internal connection of A' is in the scope of cohesion connection.

- **Merging uncoupled classes.** Let $a1, a2 \in A$ be two aspects of S . Let a' be the aspect which is the union of $a1$ and $a2$. Let S' be the AO system which is identical to S except that $a1$ and $a2$ are replaced by a' . If no relationships exist between $a1$ and $a2$, then $\text{JPCoupling}(a1) + \text{JPCoupling}(a2) = \text{JPCoupling}(a')$, $\text{JPCoupling}(S) = \text{JPCoupling}(S')$. This property states that merging two aspects that are not connected can not affect coupling in the system at all.

6. IMPLEMENTATION ISSUES

To investigate the practicability of our coupling framework, we implement a dynamic coupling measures tool called AJDCM. The tool is composed of two function modules. The first is based on collecting the coupling data from executing program into XML trace files, whereas the second calculates the measures based on XML files.

An overview of the architecture is depicted in Figure 2. The boxes of solid lines correspond to new tools, and the box of dotted lines corresponds to components of existing tools that we utilize.

The tool separates the collection and analysis of dynamic coupling measurement into two phases. In the first phase, data from a running AO program is gathered and stored in the form of XML files. This is accomplished by *instrumenting* [13] the source code running on Java Virtual Machine (JVM). In the second phase, the data is analyzed. The metric analyzer module is used to obtain the dynamic coupling measures and traverses the XML data structure into the form of HTML outcome files.

7. EMPIRICAL EVALUATION

To better evaluate dynamic coupling measures, we perform an experimental study on 12 AspectJ benchmarks to study whether dynamic coupling measures we proposed are not only theoretical but also practical in AO systems. On the other hand, we compare dynamic coupling measures with static coupling measures by

means of the analysis of dynamic pointcut designators and program statement structures to discuss whether dynamic coupling measures could more precisely reflect the coupling information for AO systems.

7.1 Subject Programs

We use 12 AspectJ benchmarks shown in Table 5 for the experimental study. Bean, Ltw, Observer, Telecom, Tjp are the examples included in the AspectJ compiler example package [4]. Example is a small working example in this paper. AJHotdraw [1], written in AspectJ, is refactored from JHotdraw [3]. The remaining programs are obtained from the abc benchmark package [2]. This group of benchmarks have also been widely used by other researchers to evaluate their work on test generation [12] and change analysis [14].

Table 5: Subject Programs

Programs	AB	Line of Code	AJ File	Java File	AJ Pro
AJHotdraw	AJHD	44494	31	290	9.66%
Bean	Bean	212	1	2	33.33%
Dcm	Dcm	2865	4	25	13.79%
Example	Exam	75	2	3	40%
Lod	Lod	2616	4	24	14.29%
Ltw	Ltw	40	1	1	50%
NullCheck	NC	2472	1	23	4.17%
Observer	OS	214	2	60	25%
QuickSort	QS	127	1	1	50%
Telecom	TC	638	3	10	23.08%
Tetris	TT	1576	8	8	50%
Tjp	Tjp	72	1	1	50%

Table 5 shows the abbreviation (AB), the number of lines of code (Line of Code), the number of AspectJ files (AJ File), the number of Java files (Java File), and the proportion of AspectJ files of all the files (AJ Pro) of the selected subject programs (Programs).

7.2 Threats to Validity

However, threats to validity should still be taken into consideration, including internal validity and external validity. There are two main aspects which are likely to cause the internal validity. Firstly, the warp of dynamic coupling measures which are calculated by AJDCM automatically may possibly occur although we have inspected the relevant source code manually. Secondly, dynamic and static coupling measures may be incompletely reasonable to form comparison pairs of coupling measures. External validity concerns the applicability and generality of our findings. 12 AspectJ benchmarks are used as the subject programs for experiments and different results may be obtained from using different programs. Even for the same programs at runtime, dynamic coupling degree may be also different.

7.3 Comparing Coupling Measures

For better comparison between dynamic coupling measures and static coupling measures, we pick out some kinds of dynamic coupling measures from this paper and some pieces of static coupling measures from our previous work [11] to form four comparison pairs of coupling measures.

7.3.1 Dynamic Coupling Measures

With respect to the set definition of dynamic measures, we find that the sets of every two measures are the subsets of the another measure. For example, IC_II and IC_IC are the subsets of IC_DC . Thus, we select four kinds of dynamic coupling measures, including $|IC_DC|$, $|IC_DM|$, $|EC_DC|$ and $|EC_DM|$.

7.3.2 Static Coupling Measures

We next choose three pieces of static coupling measures from our previous work [11], which could statically measure the similar coupling degree as dynamic coupling ones. These three static coupling measures are CDP, CAE, RFA respectively and more details are described in [11].

- **CDP (Crosscutting Degree of an Aspect caused by Pointcuts)** CDP measures the number of modules affected by the pointcuts in a given aspect.
- **CAE (Coupling on Advice Execution)** CAE measures the number of aspects containing advices possibly triggered by the execution of operations in a given module.
- **RFA (Response For a Module All)** RFA measures the number of methods and advices potentially executed in response to a message received by a given module.

7.3.3 Comparison Pairs of Coupling Measures

In terms of the descriptions of dynamic and static coupling measures, we form four comparison pairs of coupling measures. Each pair of coupling measure consists one dynamic coupling measure and one static coupling measure.

- **Pair 1. |IC_DC| and CAE**
|IC_DC| and CAE reflects import dynamic and static coupling degree of aspects triggered by the execution of operations in a given module respectively.
- **Pair 2. |IC_DM| and RFA**
|IC_DM| and RFA shows import dynamic and static coupling degree of methods and advices potentially executed in response to a message respectively.
- **Pair 3. |EC_DC| and CDP**
Similar to Pair 1, |EC_DC| and CDP measures export dynamic and static coupling degree of modules affected by the pointcuts in a given aspect respectively.
- **Pair 4. |EC_DM| and RFA**
Similar to Pair 2, |EC_DM| and RFA indicates export dynamic and static coupling degree of methods and advices potentially executed in response to a message respectively.

7.4 Experiment Results

With respect to the dynamic behaviors in AO systems at runtime, we divide comparisons of coupling measures into two parts: dynamic pointcut designators and program statement structures. By means of fine-grained analysis, dynamic pointcut designators focus on the dynamic crosscutting pointcuts in aspect code while program statement structures more concentrate on the dynamic execution of conditional and loop statements both in aspect code and base code.

7.4.1 Dynamic Pointcut Designators

With respect to the dynamic pointcut designators defined in AspectJ, we classify them into seven kinds: (1) *this* and *target*, (2) *cflow* and *cflowbelow*, (3) *perthis* and *pertarget*, (4) *percflow* and *percflowbelow*, (5) *if*¹, (6) *args* and (7) *adviceexecution*. For each kind of dynamic pointcut designator, we next compare

¹*if* in Subsubsection 7.4.1 is denoted as one kind of dynamic pointcut designators.

dynamic coupling degree with static degree in terms of the same source file.

Designator *this* and *target*

Pointcut designator *this* indicates any join point where the currently executing object is an instance of one class and *target* shows any join point where the target object is an instance of one class. Since the instance of class is created at runtime, dynamic crosscutting behavior triggered by the dynamic pointcut designators which are composed of *this* or *target*, could not be found by static analysis. Therefore, static coupling measures are unable to precisely measure the coupling degree of crosscutting behavior.

By means of source code analysis, we find that these kinds of pointcut designators are existing in most benchmarks such as Bean, Lod, Observer, Telecom, Tjp, AJHotdraw and Tetris. Table 6 shows the results of dynamic and static coupling measures in Bean, Telecom, and Tjp in which dynamic pointcuts are composed of *this* and *target*.

Table 6: Results of Dynamic and Static Coupling Measures in Bean, Telecom, and Tjp in Which Dynamic Pointcuts are Composed of *this* and *target*

FN	T	BM	ICDC	CAE	ICDM	RFA	ECDC	ECDC	CDP
BoundPoint	A	Bean	6	0	12	6	0	0	1
Point	C	Bean	0	1	0	9	12	6	0
Billing	A	TC	4	0	8	6	0	0	1
Call	C	TC	0	2	0	12	0	8	0
Connection	C	TC	0	2	0	6	6	0	0
Customer	C	TC	0	2	0	12	2	0	0
Timer	C	TC	0	0	0	3	10	0	0
TimerLog	A	TC	4	0	0	0	0	0	1
Timing	A	TC	4	1	12	2	2	4	1
Demo	C	Tjp	0	1	0	5	0	2	0
GetInfo	A	Tjp	2	0	0	6	0	0	1

Table 6 describes the filename (FN), the type (T), three static coupling measures (CDP, CAE and RFA), and four dynamic coupling measures (ICDC, ICDM, ECDC and ECDM) of each file in relevant benchmark (BM). For Column T, A is for aspect file while C is for class file. On the other hand, Column ICDC, ICDM, ECDC and ECDM measure the value of |IC_DC|, |IC_DM|, |EC_DC| and |EC_DM| respectively. In the following tables, the meanings of columns are similar to columns in Table 6.

Pair 1: For all of the aspect files, the value of |IC_DC| is larger than the value of CAE. This is because |IC_DC| calculates the number of dynamic crosscutting, which only exists in aspect files. From the viewpoint of design modularity, the aspect is less interacted with the aspect. Thus, for aspect files, CAE is normally equal to zero.

Pair 2: For most of the aspect files, the values of |IC_DM| is larger than the value of RFA, since pointcut designator *target* makes the method calls in advices, which are not counted in RFA.

Pair 3: For all of the java files, the value of CDP is not larger than the value of |EC_DC| because some dynamic behaviors could not be checked statically. Thus, |EC_DC| could more precisely reflect the coupling degree between classes and aspects than CDP.

Pair 4: For all of the aspect files, the value of RFA is larger than the value of |EC_DM|. The value of |EC_DM| is usually small when designator *this* is defined in the pointcut for the reason that it makes instances of other classes silent.

Designator *cflow* and *cflowbelow*

Designator *cflow* describes any join point in the control flow of currently join point operation while *cflowbelow* indicates any join point below the control flow. These unique features make

programmers difficult to understand the dependencies among the components in AO systems by means of static analysis. Therefore, for better understanding coupling degrees, dynamic coupling measures are indispensable dynamic pointcut designators. Table 7 distinguishes the results of dynamic and static coupling measures in Example and Lod in which dynamic pointcuts are composed of *cflow* and *cflowbelow*.

Table 7: Results of Dynamic and Static Coupling Measures in Example and Lod in Which Dynamic Pointcuts are Composed of *cflow* and *cflowbelow*

FN	T	BM	ICDC	CAE	ICDM	RFA	ECDM	ECDC	CDP
A1	A	Exam	6	0	12	3	0	0	3
A2	A	Exam	2	1	2	1	8	2	1
C1	C	Exam	0	2	0	6	0	5	0
C2	C	Exam	0	1	0	2	0	1	0
C3	C	Exam	0	0	0	1	6	0	0
EmptyQueue QException	C	Lod	0	1	0	0	0	1	0
PQ_Node	C	Lod	0	2	0	2	0	30	0
Q_Node	C	Lod	0	2	0	2	0	48	0
TQ_Node	C	Lod	0	2	0	3	0	46	0
Test	C	Lod	0	2	0	9	0	20	0
Check	A	Lod	10	1	0	4	0	0	13
ObjectSupplier	C	Lod	0	0	0	4	205	0	0
Percflow	A	Lod	79	0	149	0	0	0	27
Pertarget	A	Lod	56	0	56	7	0	0	16

For Exam, a working example in this paper, because of the designator *cflow* in A1, the values of dynamic coupling measures are obviously going up than the values of static ones. Since the working example is pretty simple and more details have been explained in Section 3, we next analyze the influence caused by designator *cflow* in Lod according to those four pairs of measures.

Pair 1: For all of the aspect files, the value of $|IC_DC|$ is much larger than the value of CAE. Since aspect file *Percflow* is declared with *percflow*, an instance of aspect is instantiated for every control flow. It makes that the aspect crosscuts many join points dynamically and the value of $|IC_DC|$ increases fast.

Pair 2: For most of the aspect files, the value of $|IC_DM|$ is extremely larger than the value of RFA except aspect file *Check*. Because aspect file *Pertarget* is declared with *pertarget*, one pointcut is instantiated with the definition of *target*. It also makes that many method calls executed in the corresponding advices and the value of $|IC_DM|$ quickly rises.

Pair 3: For all of the java files, the value of $|EC_DC|$ is obviously greater than the value of CDP. The reason is that method executions in advices which are triggered by the dynamic pointcut with *cflow* still intercept the current pointcut because of dynamic mechanism caused by *cflow*.

Pair 4: For all of the files, the value of Column ECDM is almost zero, except for *ObjectSupplier*. By checking the source code, we obtain the fact that aspect file *Percflow* and *Pertarget* are both extended from *ObjectSupplier* and there are a lot of calls to the methods of base class in aspect files.

Designator *if*

Designator *if* presents any join point where the boolean expression is true. When programs execute some join point which triggers relevant pointcut, the boolean expression must be calculated to judge whether the execution is going on or off. If the result is true, programs will run continuously. Otherwise, it will skip the pointcut and not trigger the execution of advices.

Since programmers are used to use more conditional statements in plain methods or advices, designator *if* is less used in the definition of pointcut. Of all the benchmarks, AJHotdraw and Lod have

some pieces of pointcut which is composed of *if*. Table 8 lists the results of dynamic and static coupling measures in AJHotdraw.

Table 8: Results of Dynamic and Static Coupling Measures in AJHotdraw in Which Dynamic Pointcuts are Composed of *if*

FN	T	BM	ICDC	CAE	ICDM	RFA	ECDM	ECDC	CDP
ClassInvariant	C	AJHD	0	1	0	0	3560	0	0
Invariant	A	AJHD	1810	0	3560	1	0	0	69
MockInvariant Class	C	AJHD	0	1	0	4	0	5	0
Figure Attributes	C	AJHD	0	2	0	20	0	374	0
ImageFigure	C	AJHD	0	3	0	19	0	75	0
TextFigure	C	AJHD	0	4	0	66	0	1304	0
Standard Drawing	C	AJHD	0	1	0	38	0	52	0

Although designator *if* exists in AJHotdraw and Lod, all the four pairs of coupling measures Pair 1, 2, 3, and 4 are still in the same tendency as other benchmarks.

Designator *args*

Designator *args* points out any join point where there are specific arguments. When join points are intercepted at runtime, the control flow of programs will change and pointcuts will be matched at that time. If *args* is defined as a part of the pointcuts, some accurate arguments need to be assigned. Otherwise, pointcuts will not be matched and the programs will execute orderly. Actually, designator *args* is usually coupled with designator *target* in the definition of pointcuts. Thus, all the four pairs of coupling measures Pair 1, 2, 3, and 4 are also like those relevant to designator *target*.

Designator *adviceexecution*

Designator *adviceexecution* describes the execution of all advice bodies, just like the working example in this paper. When advice a1 in A2 is triggered, pointcut pc2 intercepts and causes the execution of advice a2 in A1. For other examples which include *adviceexecution*, we could get the similar result. Note that the value of CAE for aspect files is almost zero in the above tables. However, if one aspect crosscuts another aspect, the value of CAE for the crosscutted aspect file will be more than zero. On the other hand, the four pairs of coupling measures for crosscutted aspect files are likely to retain the same direction as before while those for crosscutting aspect files are likely to increase based on original results.

Designator *wildcard*

wildcard is not one kind of dynamic pointcut designator, but it is pretty important to be used to define the variance of pointcut. However, for programmers, it is difficult to statically identify what are really matched by the pointcuts and what are not. Table 9 shows the results of dynamic and static coupling measures in NullCheck in which dynamic pointcuts are composed of *wildcard*.

Table 9: Results of Dynamic and Static Coupling Measures in NullCheck in Which Dynamic Pointcuts are Composed of *wildcard*

FN	T	BM	ICDC	CAE	ICDM	RFA	ECDM	ECDC	CDP
Simulator	C	NC	0	1	0	45	0	17	0
Scheduler	C	NC	0	1	0	11	0	3	0
PriorityQueue	C	NC	0	1	0	11	0	1	0
EnforceCoding Standards	A	NC	21	1	0	0	0	0	9

Since the *wildcard* is often used in the definition of pointcuts

in aspect files, we conceive one pair of coupling measures Pair 3 to explain the differences between dynamic coupling measures and static coupling measures. The value of CDP is statically computed and it considers all the possible matched methods. However, the number of methods executed actually is equal to zero when the programs are executed. Thus, $|EC_DC|$ is indispensable when CDP could not perfectly indicate the runtime coupling degree.

7.4.2 Program Statement Structures

On the other hand, program statement structures should be considered at runtime as well, particularly for aspect files. In order to analyze the four pairs of coupling measures more clearly, we next divide program statement structure into two parts: conditional statement and loop statement.

Conditional Statement

Conditional statements, including `if`² and `switch/case`, are often used in both aspect code and base code. When measuring static coupling measures, programmers do not consider which branch of conditional statement is really executed. However, especially for aspect files, because of the dynamic crosscutting mechanism, any branch of conditional statements is likely to have completely different influence on the results of dynamic coupling degree. Table 10 describes the results of dynamic and static coupling measures in Tetris in which there are many conditional statements. (We ran Tetris randomly for three times and the results are labelled as TT1, TT2 and TT3 respectively.)

Table 10: Results of Dynamic and Static Coupling Measures in Tetris in Which There are Many Conditional Statements

FN	T	BM	ICDC	CAE	ICDM	RFA	ECDM	ECDC	CDP
GameInfo	A	TT1	1	0	3	0	0	0	1
Menu	A	TT1	2	0	1	2	0	0	2
Counter	A	TT1	1	1	0	0	0	1	2
Levels	A	TT1	3	0	0	0	0	0	4
NewBlocks	A	TT1	80	0	0	0	0	0	3
NextBlock	A	TT1	2	1	2	0	0	5	2
TestAspect	A	TT1	8	0	0	0	0	0	1
BlockPanel	C	TT1	0	1	0	10	2	75	0
TetrisGUI	C	TT1	0	5	0	10	4	5	0
TetrisImages	C	TT1	0	1	0	5	0	8	0
Timer	C	TT1	0	1	0	5	0	1	0
AspectTetris	C	TT1	0	5	0	31	0	2	0
GameInfo	A	TT2	1	0	3	0	0	0	1
Menu	A	TT2	2	0	1	2	0	0	2
Counter	A	TT2	1	1	0	0	0	1	2
Levels	A	TT2	3	0	0	0	0	0	4
NewBlocks	A	TT2	76	0	7	0	0	0	3
NextBlock	A	TT2	2	1	2	0	0	6	2
TestAspect	A	TT2	9	0	0	0	0	0	1
BlockPanel	C	TT2	0	1	0	10	2	70	0
TetrisGUI	C	TT2	0	5	0	10	4	5	0
TetrisImages	C	TT2	0	1	0	5	7	9	0
Timer	C	TT2	0	1	0	5	0	1	0
AspectTetris	C	TT2	0	5	0	31	0	2	0
GameInfo	A	TT3	1	0	3	0	0	0	1
Menu	A	TT3	2	0	1	2	0	0	2
Counter	A	TT3	2	1	0	0	0	1	2
Levels	A	TT3	3	0	0	0	0	0	4
NewBlocks	A	TT3	103	0	89	0	0	0	3
NextBlock	A	TT3	2	1	3	0	0	7	2
TestAspect	A	TT3	9	0	0	0	0	0	1
BlockPanel	C	TT3	0	1	0	10	3	96	0
TetrisGUI	C	TT3	0	5	0	10	4	5	0
TetrisImages	C	TT3	0	1	0	5	89	9	0
Timer	C	TT3	0	1	0	5	0	1	0
AspectTetris	C	TT3	0	5	0	31	0	3	0

²`if` in Subsubsection 7.4.2 is denoted as one kind of conditional statements.

It could be seen explicitly from Table 10 that static coupling measures are always equal while dynamic measures are fluctuating. Static coupling degree measures the same source code and keeps the same result. However, dynamic coupling measures indicate different coupling degree at three different running situation.

Loop Statement

Loop statements, consisting of `for`, `while` or `do/while`, are existing in most programs. It is general to say that for each iteration, body of loop will be executed once. When the join point, which may be intercepted by pointcuts, exists in the loop body, it will trigger the execution of advices same times as the number of iteration. Table 11 lists the results of dynamic and static coupling measures in QuickSort in which there are many loop statements. (We modified the number of loop N and ran QuickSort for five times and the results are labelled as QS1 (N:1), QS2 (N:5), QS3 (N:10), QS4 (N:50), and QS5 (N:100) respectively.)

Table 11: Results of Dynamic and Static Coupling Measures in QuickSort in Which There are Many Loop Statements

FN	T	BM	ICDC	CAE	ICDM	RFA	ECDM	ECDC	CDP
Foo	C	QS1	0	1	0	4	0	2	0
QuickSort	C	QS1	0	1	0	7	0	5	0
Stats	A	QS1	7	0	0	0	0	0	2
Foo	C	QS2	0	1	0	4	0	2	0
QuickSort	C	QS2	0	1	0	7	0	46	0
Stats	A	QS2	48	0	0	0	0	0	2
Foo	C	QS3	0	1	0	4	0	2	0
QuickSort	C	QS3	0	1	0	7	0	125	0
Stats	A	QS3	127	0	0	0	0	0	2
Foo	C	QS4	0	1	0	4	0	2	0
QuickSort	C	QS4	0	1	0	7	0	1060	0
Stats	A	QS4	1062	0	0	0	0	0	2
Foo	C	QS5	0	1	0	4	0	2	0
QuickSort	C	QS5	0	1	0	7	0	2691	0
Stats	A	QS5	2693	0	0	0	0	0	2

We could easily find that the values of five groups of dynamic coupling measures are at the similar tendency. This result could clearly prove that for aspect files, the number of crosscutting is related with the number of loop.

7.5 Discussion

From all the above analysis, we find that dynamic coupling measures we proposed are reasonable and complementary to the existing static coupling measures since dynamic coupling measures could more precisely indicate the actual coupling degree of AO systems than static ones. There are several reasons for that:

- When **dynamic pointcut designators** such as `cflow`, `cflowbelow`, `if` or `adviceexecution` are composed of pointcuts, traditional static coupling measures could not precisely capture the coupling degree because of the dynamic crosscutting behavior in aspect files.
- If many **conditional statements** exist in aspect files, the values of dynamic coupling measures are likely to be change irregularly. On the other hand, if a lot of **loop statements** occur in aspect files, the values of dynamic coupling measures are high correlated with the number of iteration. However, conditional or loop statements do not have any effect on static coupling degree at all.

However, the dynamic metric values proposed in the paper may be dependent on the input data (an execution path) of a program. For each input, code coverage of current execution should be ensured to reach a reasonable value, so as to obtain the useful data.

On the other hand, the dynamic metric values might be correlated with the implementation detail of a program as well. Thus, for better gathering the precise values of dynamic coupling metrics, static and dynamic preliminary analysis should be performed efficiently.

8. RELATED WORK

In this paper, the dynamic coupling measurement for AO software is on the basis of Arisholm's framework [5] by taking into account AO specific composition models and frameworks about AO static coupling measurement proposed in [6, 7, 15].

Zhao [15] implicitly extends the framework by formally describing new forms of dependencies between aspects and classes. Although it intends to be a generic framework for AO languages, it does not take into consideration a number of relevant AOP mechanisms, such as coupling between the aspects themselves.

Bartsch and Harrison [7] explicitly extend the framework proposed by Briand *et al.* [8] aiming at AspectJ. They focus on describing new types of connections and analyzing the impact on other criteria. They have also created a new criterion specific for AspectJ, which was not available on the framework carried out by Briand *et al.* [8]. This paper's limitation is that the target is an specific language instead of a generic language independent. And also, we find out that some problems are deserved discussing in this paper. Bartsch and Harrison [7] propose that if the current object and the target object of a join point are of different classes, the join point will be associated with both classes. In this paper, with regard to dynamic coupling measurement, current object is considered as having coupling connection associated with the join point.

The main goal of work carried out by [6] is to create a framework which is generic enough to be language independent, but concrete enough to be easily instantiated for any AO language. They briefly present dynamic issues that affect the creation of coupling measures without a deep analysis.

We realized that an extension of the framework proposed by Arisholm [5] is not enough to cover new AO composition mechanisms and features which was not present on other works. Hence, our approach was to analyze the new composition mechanisms and abstractions introduced by AO programming languages and, after a deep understanding of their properties and relationships, create a dynamic coupling framework, from which we derive our measures. In addition, we empirically compare dynamic coupling measures with static ones by means of fine-grained analysis: dynamic point-cut designators and program statement structures.

9. CONCLUSIONS AND FUTURE WORK

With the emergence of AO programming languages and its advanced composition mechanism for promoting enhanced separation of crosscutting concerns, the work dedicated to different research fields in AO systems becomes popular.

This paper has presented a framework for dynamic coupling measurement for AO software. This framework offers a sound base on which existing aspect-oriented measures can be analyzed and has been applied to formally define 12 measures that targets the control flow changes caused by aspects. The mathematical properties of measures are validated. On the other hand, empirical evaluation of dynamic coupling measures is performed. As a result, we find that dynamic coupling measures we proposed are complementary to the existing static ones since dynamic coupling measures could reflect the real coupling dependencies of modules in AO systems.

In the future work, we intend to proceed our work in the following directions. Firstly, explore the relationship between the inheritance of aspects and classes and methodology of dynamic coupling

measurement for AO systems. Secondly, collect data from various industries to conduct more empirical validation. Thirdly, build a model to predict the quality of a system using statistical analysis.

10. REFERENCES

- [1] AJHotDraw. <http://sourceforge.net/projects/ajhotdraw/>.
- [2] The AspectBench Compiler. <http://abc.comlab.ox.ac.uk/>.
- [3] JHotDraw. <http://sourceforge.net/projects/jhotdraw/>.
- [4] The AspectJ Team. The AspectJ Programming Guide. Online manual, 2003. <http://eclipse.org/aspectj/>.
- [5] E. Arisholm, L. Briand, and A. Foyen. Dynamic coupling measurement for object-oriented software. *Software Engineering, IEEE Transactions on*, 30(8):491–506, 2004.
- [6] T. Bartolomei, A. Garcia, C. Sant'Anna, and E. Figueiredo. Towards a unified coupling framework for measuring aspect-oriented programs. *Proceedings of the 3rd international workshop on Software quality assurance*, pages 46–53, 2006.
- [7] M. Bartsch and R. Harrison. A Coupling Framework for AspectJ. *Extended Abstract, To appear in Proc. Empirical Assessment of Software Engineering Conference-EASE*, 6, 2006.
- [8] L. Briand, J. Daly, and J. Wust. A unified framework for coupling measurement in object-oriented systems. *IEEE Transactions on Software Engineering*, 25(1):91–121, 1999.
- [9] M. Ceccato and P. Tonella. Measuring the Effects of Software Aspectization. *Proceedings of the 1st Workshop on Aspect Reverse Engineering (CD-ROM), The Netherlands*, 2004.
- [10] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, and W. Griswold. An Overview of AspectJ. *Ecoop 2001-Object-Oriented Programming: 15th European Conference, Budapest, Hungary, June 18-22, 2001: Proceedings*, 2001.
- [11] H. Shen and J. Zhao. An evaluation of coupling metrics for aspect-oriented software. Technical Report SJTU-CSE-TR-07-04, Center for Software Engineering, SJTU, Apr 2007.
- [12] T. Xie and J. Zhao. A framework and tool supports for generating test inputs of aspectj programs. In *In AOSD '06: Proceedings of the 5th international conference on Aspect-oriented software development*, pages 190–201, New York, NY, USA, 2006. ACM Press.
- [13] K. Yeung, P. Kelly, and S. Bennett. DYNAMIC INSTRUMENTATION FOR JAVA USING A VIRTUAL JVM. *Performance Analysis and Grid Computing*, 2004.
- [14] S. Zhang and J. Zhao. Change impact analysis for AspectJ programs. Technical Report SJTU-CSE-TR-07-01, Center for Software Engineering, SJTU, Jan 2007.
- [15] J. Zhao. Measuring Coupling in Aspect-Oriented Systems. *Proc. of the 10th International Software Metrics Symposium (METRICS)*.

Acknowledgments

This work was supported in part by National High Technology Development Program of China (Grant No. 2006AA01Z158), National Natural Science Foundation of China (NSFC) (Grant No. 60673120), and Shanghai Pujiang Program (Grant No. 07pj14058). We would like to thank Sai Zhang for his discussion on this work.